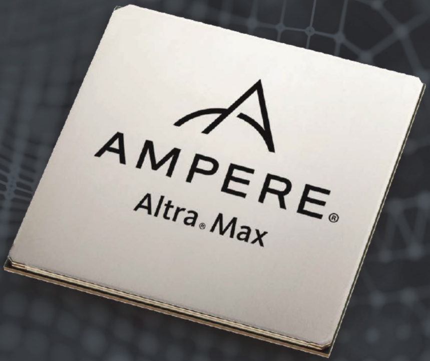
# A Performance Analysis Methodology for Optimizing Ampere® Altra® Family Processors



## Overview

This tutorial describes a hierarchical performance analysis methodology that enables developers to deliver high performance applications on Ampere® Altra® and Ampere® Altra® Max 64-bit multi-core processors. Using a systematic approach to identify performance bottlenecks is key to successful performance analysis. We recommend a hierarchical methodology to first understand the high-level performance characteristics and then explore lower-level details as needed.

This methodology starts by measuring platform level performance to determine if the platform can effectively run the application. If platform level bottlenecks exist, determine if these can be eliminated by hardware changes. The platform level analysis highlights lower-level performance data to collect. This saves time by highlighting the lower-level details that should be investigated instead of spending time collecting and analyzing a lot of low-level details that may or may not be relevant.

This tutorial is written for Application Architects who wish to learn the methodology to effectively characterize and improve application performance on Ampere Altra Family processors.

## A Systematic Approach to Performance Analysis

Amdahl's Law is a cardinal rule to consider when embarking on performance analysis and optimization. When applied at a high level, Amdahl's Law suggests focusing on areas or applications that, if improved, would have the most impact. Widely used critical applications, for example, merit more detailed performance analyses because of the potential for wide-ranging impact. It is worth noting that while guidelines help, experienced performance engineers expect the unexpected, question all results to make sure they are self-consistent and not caused by performance tools or measurement methodology. It is especially important to question any assumptions.

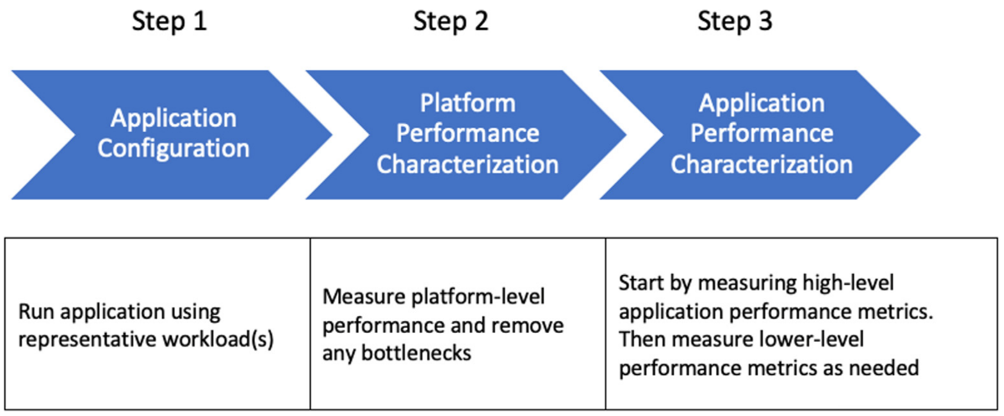
Figure 1 shows the steps in the hierarchical performance methodology described in this tutorial. It is important to note that often this involves an iterative process of characterizing performance, making code changes that are expected to improve performance, and collecting the performance data using the same inputs to determine if a change improves performance. If the changes reduce the previous performance bottleneck, there are probably new performance bottleneck(s) to be investigated. Over time, the iterative nature of performance analysis and optimization becomes apparent.

The first step is the choice of workload or application. In general, workloads being studied should have these attributes:

- Measurable – the workload must have a well-identified metric. To improve something, you must be able to measure it.
- Repeatable – the workload should report the same (within a margin of error) performance regardless of how many times it is run.
- Representative – above all, the workload should represent the production application. Improvements made to the workload should translate to similar improvements in the real world. Modern applications execute very different code paths depending on inputs, so the choice of inputs is equally important.

While these steps might appear trivial, they are critical to the overall performance analysis effort.

Figure 1: Steps in the Hierarchical Performance Methodology


Step 1
Step 2
Step 3
Application Configuration
Platform Performance Characterization
Application Performance Characterization
Run application using representative workload(s)
Measure platform-level performance and remove any bottlenecks
Start by measuring high-level application performance metrics. Then measure lower-level performance metrics as needed

The second step of the hierarchical performance methodology is to characterize the platform to determine if the platform can effectively and efficiently run the application before diving into lower-level and application-specific performance metrics. A trivial example is an application that requires more memory than is available on the platform, thrashing the system by constantly swapping virtual memory to and from a storage device. For this example, jumping directly to fixing low level performance problems such as cache misses would not result in expected improvements while the overarching issue of insufficient memory capacity exists. This step in the hierarchical performance analysis methodology highlights what additional information is useful to collect, and, just as importantly, what isn't relevant.

We recommend characterizing CPU, memory, network utilization, and Operating System (OS) performance metrics to understand if the platform limits application performance. Certain applications may stress a particular platform subsystem, which limits application performance. In these cases, upgrading the specific platform component can have a big impact on improving performance.

Many performance tools can measure various areas of performance, and many of the tools overlap in function. Because some tools are easier to use and summarize data more conveniently, we recommend that you find the tools that work best for you and learn how to use them. The important thing is to first characterize performance at the platform level. Table 1 lists the common, system-level performance tools and the metrics they report.

Table 1: System Level Tools and Reported Metrics

| CHARACTERIZED SUBSYSTEM | TOOLS | METRICS |
|---|---|---|
| CPU | sar, mpstat | CPU Core utilization, user time, kernel time |
| Memory | sar, vmstat | Memory usage, paging statistics |
| OS | sar, vmstat | Task creation, scheduling, context switches, OS run queues |
| Storage IO | sar, iostat | Disk bandwidth, IO Operations per Second (IOPS) |
| Network IO | Iftop, iperf3, sar | Network bandwidth, latencies |

The third step in the hierarchical performance methodology is to characterize application performance. In step two, we measured high level platform performance metrics. Based on what was discovered in the second step, collect additional data for a given subsystem that has performance bottlenecks as well as collecting application performance data. Table 2 lists some tools that can be used in this step.

Table 2: Tools and Metrics for Characterizing Application Performance

| TOOLS | METRICS |
|---|---|
| htop, mpstat, sar | Measures how effectively CPU cores are being utilized. Is the application CPU bound or is a system level bottleneck limiting performance? |
| perf stat | Gives real-time insight into application & kernel functions and how they affect performance |
| perf record | Allows drilling down into the application to measure lower-level performance data from the application level all the way down to individual lines of source code. |

perf stat and perf record are powerful tools that can collect large amounts of profile data from performance counters in the hardware. Two of the most important performance counters are instructions retired and cycles. Cycles show what is taking the most CPU time, and the instructions per cycle (IPC) ratio shows how efficiently the application runs on the processor.

A "good" IPC value depends on what an application is doing, so detailed advice on what constitutes a good value is beyond the scope of this document. The article "Arm Neoverse N1 – Performance Analysis Methodology to Tune Production Systems and Application Code," available here, provides detailed information about using perf to collect performance metrics.

## Key Findings and Conclusions

This tutorial describes a tried-and-tested hierarchical performance analysis methodology to characterize applications running on Ampere Altra processors. Starting with workloads that are measurable, repeatable, and representative is an important first step to a successful performance analysis. It is important to study the system holistically; first measure platform level performance, which the performance community at Ampere has used extensively to identify and fix platform level performance issues. After understanding the platform level performance, application performance data should be collected with a focus on bottlenecks identified by the platform level study. Additionally, we provide a reference to learn more about effective use of performance tools.